DOCUMENT RESUME

ED 233 687 IR 010 777

AUTHOR Hecht, Herbert
TITLE The Introduction of Software Tools. Final Report.
INSTITUTION SoHaR Inc., Los Angeles, CA.
SPONS AGENCY National Bureau of Standards (DOC), Washington, D.C. Inst. for Computer Sciences and Technology.
REPORT NO NBS-SP-500-91
PUB DATE Sep 82
NOTE 43p.
AVAILABLE FROM Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402 (1982-360-997/2207, $4.75).
PUB TYPE Guides - General (050) -- Reports - Research/Technical (143)

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS Check Lists; *Computer Programs; Computer Science; Guidelines; Management Information Systems; *Man Machine Systems; Occupational Information; *Program Implementation; Programing; *Public Agencies; Purchasing; *Selection
IDENTIFIERS Scientific Computer Programs; *Software Evaluation; *Software Tools

ABSTRACT

This publication provides guidance for the introduction of computer software tools within agencies of the United States government or within other installations where there has been little or no prior use of software tools. From a survey of current tool usage, it is concluded that the greatest obstacles to effective use of software tools are encountered in organizations employing fewer than 40 programmers, and the needs of these environments are therefore emphasized. Specific needs for software tools in programming for management information systems (MIS) and for scientific applications are discussed. Measures are described to overcome organizational obstacles to use of tools, to deal with problems arising from the tools, and to reduce the difficulties posed by existing computer installations. Also described are the responsibilities of software management, software engineers, and toolsmiths (who make minor modifications to tools and the computer environment as necessary). Steps required for the successful introduction of tools are then presented, organized both by the function responsible for their accomplishment, and the time schedule in which they must be completed. Work to be performed in each step is described in detail. A list of 14 references and the agenda and attendance list from a related workshop conclude the publication. (Author/ESR)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED233687

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☑ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

Computer Science and Technology

NBS Special Publication 500-91

The Introduction of Software Tools

Herbert Hecht

SoHaR Incorporated
1040 So. La Jolla Avenue
Los Angeles, California 90035

U.S. DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary

National Bureau of Standards
Ernest Ambler, Director

Issued September 1982

IR01047?

2

4.75

NATIONAL BUREAU OF STANDARDS

The National Bureau of Standards[1] was established by an act of Congress on March 3, 1901. The Bureau's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the Nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau's technical work is performed by the National Measurement Laboratory, the National Engineering Laboratory, and the Institute for Computer Sciences and Technology.

THE NATIONAL MEASUREMENT LABORATORY provides the national system of physical and chemical and materials measurement; coordinates the system with measurement systems of other nations and furnishes essential services leading to accurate and uniform physical and chemical measurement throughout the Nation's scientific community, industry, and commerce; conducts materials research leading to improved methods of measurement, standards, and data on the properties of materials needed by industry, commerce, educational institutions, and Government; provides advisory and research services to other Government agencies; develops, produces, and distributes Standard Reference Materials; and provides calibration services. The Laboratory consists of the following centers:

Absolute Physical Quantities[2] — Radiation Research — Chemical Physics — Analytical Chemistry — Materials Science

THE NATIONAL ENGINEERING LABORATORY provides technology and technical services to the public and private sectors to address national needs and to solve national problems; conducts research in engineering and applied science in support of these efforts; builds and maintains competence in the necessary disciplines required to carry out this research and technical service; develops engineering data and measurement capabilities; provides engineering measurement traceability services; develops test methods and proposes engineering standards and code changes; develops and proposes new engineering practices; and develops and improves mechanisms to transfer results of its research to the ultimate user. The Laboratory consists of the following centers:

Applied Mathematics — Electronics and Electrical Engineering[2] — Manufacturing Engineering — Building Technology — Fire Research — Chemical Engineering[2]

THE INSTITUTE FOR COMPUTER SCIENCES AND TECHNOLOGY conducts research and provides scientific and technical services to aid Federal agencies in the selection, acquisition, application, and use of computer technology to improve effectiveness and economy in Government operations in accordance with Public Law 89-306 (40 U.S.C. 759), relevant Executive Orders, and other directives; carries out this mission by managing the Federal Information Processing Standards Program, developing Federal ADP standards guidelines, and managing Federal participation in ADP voluntary standardization activities; provides scientific and technological advisory services and assistance to Federal agencies; and provides the technical foundation for computer-related policies of the Federal Government. The Institute consists of the following centers:

Programming Science and Technology — Computer Systems Engineering.

[1]Headquarters and Laboratories at Gaithersburg, MD, unless otherwise noted; mailing address Washington, DC 20234.
[2]Some divisions within the center are located at Boulder, CO 80303.

Reports on Computer Science and Technology

The National Bureau of Standards has a special responsibility within the Federal Government for computer science and technology activities. The programs of the NBS Institute for Computer Sciences and Technology are designed to provide ADP standards, guidelines, and technical advisory services to improve the effectiveness of computer utilization in the Federal sector, and to perform appropriate research and development efforts as foundation for such activities and programs. This publication series will report these NBS efforts to the Federal computer community as well as to interested specialists in the academic and private sectors. Those wishing to receive notices of publications in this series should complete and return the form at the end of this publication.

National Bureau of Standards Special Publication 500-91
Natl. Bur. Stand. (U.S.), Spec. Publ. 500-91, 41 pages (Sept. 1982)
CODEN: XNBSAV

Library of Congress Catalog Card Number: 82-600577

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1982

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402
Price $4.75
(Add 25 percent for other than U.S. mailing)

ABSTRACT

From a survey of current tool usage it is concluded that the greatest obstacles to effective use of software tools are encountered in organizations employing fewer than 40 programmers, and the needs of these environments are therefore emphasized. Specific needs for software tools in programming for management information systems and for scientific applications are discussed. Measures are described to overcome organizational obstacles to use of tools, to deal with problems arising from the tools, and to reduce the difficulties posed by existing computer installations.

Steps required for the successful introduction of tools are organized in two ways: by the function responsible for their accomplishment, and by the time schedule in which they must be completed. The detail work to be performed in each step is described.

Key words: computer environments; software; software engineering; software management; software quality; software tools; toolsmith.

TABLE OF CONTENTS

		PAGE
1.	EXECUTIVE SUMMARY	1
2.	INTRODUCTION	3
3.	CHARACTERIZATION OF USER ENVIRONMENTS	5
3.1	Classification of Environments	5
3.2	Selection of Target Environments	6
3.3	The Smaller MIS Environment	8
3.4	The Smaller Scientific Environment	10
4.	USER TOOL NEEDS	12
4.1	Organizational Factors in Tool Needs	12
4.2	Application Factors in Tool Needs	16
4.3	Needs of Other Environments	19
4.4	Resources for Tool Selection	20
5.	DEVELOPMENT OF EVENT SEQUENCES	21
5.1	Purpose of Event Sequences	21
5.2	Recommended Event Sequence	24
	REFERENCES	34
	APPENDIX - WORKSHOP ON PHASING OF SOFTWARE TOOLS	35

SECTION 1

EXECUTIVE SUMMARY

This publication is intended to provide guidance for the introduction of software tools for agencies of the U. S. Government and for computer users at large. It is primarily aimed at installations where there had been little or no use of software tools previously. In a survey of software tool usage it was found that the size of the programming group had a significant effect on the extent of tool usage, with organizations of less than 40 programmers much less likely to be tools users. To provide help to these smaller organizations in the introduction and use of software tools is therefore one of the goals of this document.

Difficulties in the introduction of tools can arise in three areas:

Organizational obstacles.
Problems arising from the tools.
Obstacles in the computer environment.

Organizational obstacles can be reduced if a responsible management level is involved in the introduction of tools. Those who commit the resources for tool acquisition and use should participate actively in the relevant decisions. Their involvement in the following is particularly important:

1. Identifying the goals to be met by the tool (or by the technique supported by the tool), and assigning responsibility for the activities required to meet these goals.

2. Approving a detailed tool acquisition plan that defines the resource requirements for procurement and in-house activities.

3. Approving procurement of tools and training if this is not explicit in the approval of the acquisition plan.

4. Determining after some period of tool use whether the goals have been met.

Problems arising from the tools can be avoided by a careful, methodical selection of tools. In particular, distinct contributions to the tool selection are specified for software management and the software engineer. Software management is assigned responsibility for:

1. Identifying tool objectives.

2. Approving the acquisition plan (higher approvals may also be required).

3. Defining selection criteria.

4. Making the final selection of the tool or the source.

The software engineer is responsible for:

1. Identifying candidate tools.

2. Applying the selection criteria or preparing technical sections for a Request for Proposals (RFP).

3. Preparing a ranked list of tools or sources.

Further, the ultimate user of the tool should be involved in reviewing either the list of candidate tools or, for formal procurement, the tool requirements.

Obstacles in the computer environment are primarily due to the great diversity of computer architectures and operating system procedures, and to the lack of portability of most software tools. Activities associated with the introduction of tools can only modestly alleviate these difficulties. Guidance is provided for:

1. A methodical process of identifying candidate tools and selecting among these on the basis of established criteria, including a definition of the computer interface. This will avoid some of the worst pitfalls associated with "borrowing" a tool from an acquaintance or procuring one from the most accessible tool vendor.

2. The assignment and training of a toolsmith who can make minor modifications to both the computer environment and the tool. This is expected to provide relief where there are version-related or release-related incompatibilities with the operating system, or where the memory requirements of the tool exceed the capabilities of the installation. In the latter case, remedies may be provided by removing tool options or by structuring the tool program into overlays.

As part of this work, an event sequence for the introduction of tools has been developed that identifies specific tasks, the assignment of responsibilities for the tasks, and the order in which they have to be carried out.

SECTION 2

INTRODUCTION

This publication is intended to provide guidance for the introduction of software tools for agencies of the U. S. Government and for computer users at large. It is primarily aimed at installations where there had been little or no use of software tools previously. In a survey of software tool usage it was found that the size of the programming group had a significant effect on the extent of tool usage, with organizations of less than 40 programmers much less likely to be tools users [HECH81]. In particular, organizations of less than 40 programmers were found to need help in order to acquire and employ software software tools successfully, and the requirements of these organizations are given special emphasis.

Many of the difficulties reported by novice users with software tools can be overcome by systematic practices in the selection, acquisition, and preparation for use of software tools. This report first derives the need for specific guidance in the introduction of tools by examining a number of programming environments, and then describes the practices suited to these environments.

Section 3 charaterizes user environments in terms significant for the introduction of software tools. In this characterization, two environments were identified that will benefit most from formal guidance for the introduction of tools, and a vignette of each of these is presented in the final parts of Section 3.

Tool needs for various user environments are described in Section 4. First, a fairly broad discussion of organizational and application factors that govern tool needs is presented. Then, based on these considerations, a generic (features based) identification of tool needs for the two target environments is made. Needs of other environments are also discussed, and special attention is focused on the integration of tools. The final part of Section 4 covers resources for the selection of tools. The recent publication of a report on software development tools by NBS/ICST is of major assistance in this area [HOUG82]. The generic software tool nomenclature used in the present report is taken from [HOUG81] which in turn incorporates major portions of a Software Tool Taxonomy [REIF80].

The time phasing aspect of the introduction of tools is described in Section 5 by means of event sequences. The purpose of event sequences is discussed in general terms, and the specific event sequence for the introduction of software tools into the smaller programming environments is then developed. The events are classified by area of responsibility and precedence relationships. and each of the required events is described in detail.

A preliminary draft of this document was discussed at a Workshop on Phasing of Software Tools which was held at NBS on 18 May 1981. The agenda and the attendance list are reproduced in the Appendix. The participants contributed

many constructive comments which have been incorporated into the present version. Written comments were received from several individuals who could not attend the workshop; these contributors have been listed as "reviewers" in the Appendix.

The author wishes to acknowledge the contributions of collaborators in the preparation of this document. Myron Hecht analyzed the survey results which form the basis for Section 3, and Donald J. Reifer classified tool needs as reported in Section 4. Much helpful guidance in the conduct of this study was received from the technical monitor for the contract, Mr. R. C. Houghton, Jr. Continued encouragement and many helpful suggestions were furnished by Dr. Martha Branstad, the ICST Software Quality Program Manager.

SECTION 3

CHARACTERIZATION OF USER ENVIRONMENTS

This section considers the characterization of user environments along lines that are significant for the introduction of software tools. The starting point for this characterization is the classification of software tool users which is summarized in subsection 3.1. The selection of target environments for the introduction of software tools, based on this classification, is described in subsection 3.2. The smaller classes of management information system (MIS) and scientific programming environments are identified as most in need of outside assistance in tool usage, and vignettes typical of each of these environments are presented in subsections 3.3 and 3.4, respectively.

3.1 CLASSIFICATION OF ENVIRONMENTS

A Survey of Software Tools Usage [HECH81] considers the effect on tool usage of a fairly large number of environmental factors, including:

Size of software organization.
Type of organization (private, Government-support, Government).
Applications (scientific, MIS) and language.
Development environment (batch, interactive).
Program running environment (batch, interactive, real-time).
Computer type.
Involvement in tool development.

The first and last factors were found to have a significant effect on the extent of tool usage. The type of organization was not found to be a major determinant of the extent of tool usage in this survey. The other factors had some effect on the types of tools that were used but not on the extent of tool usage (or the effect was masked by correlation with primary determinants of tool usage).

In the following discussion the extent of tool usage is classified into three levels:

Level 0 Minimal tool usage - only tools normally provided with the operating system were in use (assemblers, loaders, compilers, debug aids, and interpreters).

Level 1 Intermediate tool usage - special purpose tools suited for the mission of the organization but without explicit effect on software quality were in use. Examples are simulators, file managers, and elementary precompilers.

Level 2 General purpose tool usage - general purpose tools, involving static and dynamic analysis features, were deliberately acquired or developed in order to enhance software quality and productivity. This group represents the highest level of tool utilization identified in the survey.

By interpreting the level index (0, 1, or 2) as a number, an average level of tool utilization can be computed for groups of tool users. The average level of tool utilization as affected by the size of the organization is shown in Table 3-1.

TABLE 3 - 1 LEVEL OF TOOL UTILIZATION

Size of Organization	Avg. Level of Tool Utilization
Small - up to 14 programmers	0.8
Medium - 15 to 39 programmers	0.8
Large - 40 to 99 programmers	1.4
Very large - over 100 programmers	2.0

The term programmer includes analysts, programming supervisors, and programming trainees but not computer operators, librarians, or other support personnel. The above data are based on a survey of 22 organizations. Tool developers were not included in this population.

3.2 SELECTION OF TARGET ENVIRONMENTS

As can be seen from Table 3-1, the use of general purpose software tools was considerably less prevalent among small and medium software organizations than among the large and very large organizations. In all size classifications there was representation of private, Government, and Government-support organizations (the three classifications for type of organization considered in this study). No evidence was found that the organization type affects the level of tool usage, but because of the small sample size this is regarded as only a tentative conclusion.

These data indicate that small and medium software organizations will represent the target environment that stands to benefit most from the availability of a comprehensive methodology for the introduction of software tools. In addition to the low level of current tool usage shown in Table 3-1, the following factors indicate that small and medium organizations need outside assistance in the introduction of tools:

1. Their awareness of tools in general, and their knowledge about specific tools suited to their needs, are frequently much less than that of larger organizations.

2. Their knowledge of tool acquisition and installation practices tends to be inadequate to permit them to obtain the full benefit from available tools.

3. Even when suitable tools are obtained and installed, these organizations frequently cannot mobilize the resources required for optimum tool utilization, such as training, start-up efforts, and change in practices to fully utilize a tool.

A further consideration (which partly encompasses all of the above) is that a given level for effort in developing a methodology for the introduction of tools can be expected to provide much more significant and measurable results if that effort is targeted at organizations at the smaller end of the size range.

The above does not imply that large, and even very large, organizations cannot benefit from further developments of methodology for the introduction of software tools, and specifically from efforts in that area undertaken by NBS/ICST. The needs of these environments are further addressed in subsection 4.3 of this report.

The need for outside assistance for the development of a suitable introduction methodology is shared by small and medium size organizations. There are only minor differences in the details of the application of the methodology between small and medium size organizations, and to avoid long titles the term "smaller" will henceforth designate the two groups collectively. Within the smaller size groups, the introduction methodology will focus on Government organizations although, as will be explained shortly, most of the introductory practices are not expected to vary significantly as a function of the organization type. The reasons for focusing on Government organizations are:

1. The demand for uniformity of software practices in Government agencies is expected to increase, and tools can be of assistance in providing and enforcing this uniformity. Hence, a greater need for tools is expected to arise in this environment.

2. Government agencies usually have a greater need to control procedural aspects of software development, and many tools address that need very specifically.

3. There are a large number of tools currently in Government inventory, and some of these are resident on computers that can be accessed by other Government organizations via terminals. Experience with tools may be shared, and help with tool problems may be furnished more readily among Government agencies than within the private sector or between Government and private organizations. Thus, the opportunity for tool usage is greater among Government organizations.

4. Successful use of a tool in a Government organization is likely to become generally known (via professional organizations, computer user groups, etc.) whereas smaller private organizations may wish to restrict the dissemination of this information for competitive reasons. Thus, the ripple effect can be expected to be greater if Government organizations are addressed as the primary target for the tool introduction methodology.

Except for the factors mentioned above, the activities and level of effort required for the introduction of tools are not believed to be significantly different among private. Government-support, and Government organizations. The greater availability of tools may appear to confer a material advantage on Government organizations but at present this has not been a cause for increased usage. The annual licensing fee for a typical tool is of the order of $ 1,000, and purchasing costs are five to ten times that amount. These are usually not the dominant expenses in the introduction of a software tool. A large number of tools are in the public domain and copies can be obtained at nominal cost from computer vendors, universities, and some organizations which specialize in this field. Thus, although the terminology used in the following may be specific to Government organizations, the general concepts are believed to be broadly applicable.

Among the smaller Government organizations, the survey found differences in tool needs that indicate that administrative and scientific environments may best be treated separately for some aspects of the introduction of software tools. A demonstrable difference is in the types of tools needed (in turn dictated by the languages used): the most widely encountered tool in smaller scientific organizations is a FORTRAN preprocessor. whereas COBOL environments frequently use optimization tools that have no direct counterparts in the scientific environments. A more subtle difference exists in the overall attitude towards tools. Scientific programmers (specifically engineers and scientists doubling as programmers) know about tools and may be conscious of some of the advantages that they confer, but are interested primarily (sometimes exclusively) in solving scientific or engineeering problems. They are only slightly motivated to devote any effort toward the enhancement of software quality. Programmers and first level supervision in the smaller administrative or MIS (Management Information Systems) environment may be only vaguely aware of tools but are highly motivated to improve the quality of their software, particularly its maintainability.

The following subsections provide vignettes of the smaller MIS and scientific environments, respectively, that particularly emphasize factors pertinent in the introduction of tools.

3.3 THE SMALLER MIS ENVIRONMENT

The term MIS environment is intended to include all programming for fiscal, administrative, housekeeping, and record-keeping functions. The predominant language is COBOL but a fair amount of assembly language programming (in application programs) is also in use. ALGOL and PL/1 are used occasionally in a few agencies. Practically all system programs used by the smaller MIS organizations are written in assembly language.

By our definition, a smaller organization may include up to 39 programmers, but the representative Government organization in this category rarely involves more than 25 or 30 programmers. It is typically a field office or a central programming organization for a specialized agency or function within an agency. There are two levels of supervision. The lower one deals with a specific

programming area (systems, disbursals, security, etc.) while the major responsibility of the upper supervisor is to maintain liaison with the headquarters organization which generates the requirements and funding for the office. Very few, if any, of the smaller Government MIS organizations can make it a major assignment for one of their employees to provide guidance in software technology and programming practices. Some of this guidance might be provided by headquarters organizations, and thus will be relayed through the highest supervisory level. But without a specific local designee who provides follow-up, much of the impact of headquarter guidance will be lost.

The range of programmer skill levels encountered in MIS organizations is broader than that prevalent in scientific environments, primarily due to the use of programmer trainees by the MIS organizations. The formal training of the programming trainees consists of in-house courses, technical school courses, and approximately 1 year of attendance at a community college. They are trained for program writing rather than software design or broader aspects of computer science or software engineering. There is also little involvement in standards or professional activities among the MIS organizations. Indicating few opportunities for a continuing, broadening education of the programmers in this environment.

The primary activity of the smaller MIS organizations frequently is program maintenance. The programs undergo almost constant change due to:

Changes in legislation.

Changes in administrative procedures.

Major organizational restructuring.

Program or functional improvement.

Correction of errors.

Offices have backlogs that range up to 1 year. Maintenance is a slow and difficult process because of the lack of good documentation (a factor that transcends this environment), the low skill levels, and the lack of good tools. When available, tools may be used very effectively, e. g., the use of a file manager for configuration management of the programs, or full employment of the features of a sophisticated editor.

In general, the smaller Government MIS organizations do not lack motivation for tool use and make use of available tools. Frequently, however, they lack both the knowledge and the resources to use tools more effectively. They will benefit from outside assistance in all of the areas identified in 3.2 above.

As far as tool needs are concerned, the MIS environment presents some unique problems, e. g., the lack of portability of most COBOL programs. and the run-time inefficiencies caused by most commercial COBOL compilers. The first problem prevents agencies from sharing application programs. even where the purpose served and the records to be generated are identical, unless they also have the same computer. The lack of portability is more of an obvious problem

In this environment than in the scientific one because many applications are common to practically every business environment: payroll, budgeting physical asset management, and billing. Among Government agencies there are further commonalities due to Government regulations, interaction with the General Services Administration, and requirements of the Office of Management and Budget. In addition to the obstacles which the lack of portability represents to the interchange of programs, it also creates great problems if a computer is being replaced by a more capable model from a different manufacturer. Conversion activity due to replacement of a central computer complex can extend over several years and represent resource expenditure comparable to the hardware cost.

The run time inefficiencies of COBOL compilers could be tolerated in the past (at least in some cases) when computer programs were used to generate periodic hard copy reports which would then serve as the user's primary data base. The predominant practice today is to update these reports continuously and to make them available to the user on interactive terminals rather than in printed format. This puts a much higher premium on efficient execution of programs because of the more frequent access and the need for a rapid response to user requests. To meet the demands of the current environment, either more computers have to be installed or the efficiency of the object code has to be improved. The latter approach has some obvious limitations, but within these it is a much more cost-effective way of improving the performance of a computer complex. Optimization programs of several types are available to deal with this problem. These are not commonly in use in smaller organizations of any type but they are frequently encountered in larger MIS organizations.

3.4 THE SMALLER SCIENTIFIC ENVIRONMENT

The typical size of the smaller scientific software development group is also 25 to 30 programmers, and two levels of supervision are involved. The lower supervisory level tends to be application oriented but the top supervisory function operates much more autonomously than in equivalent MIS agencies. Though constrained by budgets that are determined at a higher managerial level, the second level scientific supervisor typically assumes full responsibility for the technology employed within his or her organization. Where this supervisor takes an interest in software technology, structured programming supported by appropriate tools is likely to be used. On the other hand, if the interest of the supervisor is confined to a scientific specialty (simulations, engineering analysis), software technology can be a very low priority item.

Most programmers in the smaller scientific environment have an engineering or science degree but their formal training in programming may not be very advanced. Frequently, it consists of undergraduate computer or programming courses, supplemented by on-the-job training and an occasional extension course. In some groups at least one individual has a degree in computer science or a related field. The motivation of individual programmers is governed by the needs of their application. In the simulation field, which represents a significant part of the smaller scientific programming community, the code tends to be bound to a specific facility. Although most programming is in FORTRAN, there may be frequent recourse to assembly routines to speed up the execution.

As a rule, structured programming is not used in that environment although individual programmers may be experimenting with it.

Engineering and scientific analysis programs are sometimes distributed outside the originating organization, and in those cases portability is a recognized requirement, at times enforced by a portability analyzer. Structured and modular programming is used more frequently than in the simulation field but is seldom formally required. Another characteristic of the engineering or scientific analysis environment that affects tool usage is that many programming tasks are of less than 3 months duration, and much of that time is spent on analysis of the underlying problem rather than on program design or implementation. This discourages the use of tools that require much set-up time or a lengthy learning period.

The emphasis in the scientific programming environment is more on the generation of new programs as contrasted with maintenance. Some maintenance activities, such as the addition of a major feature to a simulation, or the extension of the capabilities of an analysis program, are regarded as creative and desirable assignments. More typical maintenance activities, such as modifying a report format, adapting a program to a change in hardware configuration, or correcting interface problems are regarded as less desirable assignments and are given to junior personnel as "training".

Supervisors regard the documentation and detailed maintenance as problem areas. Although these are recognized as essential elements of the organization's overall assignment, the senior programmers take little interest in them, and a good methodology is not available for breaking them down into tasks that could be efficiently handled by less experienced personnel or by personnel not involved in the direct programming. Some smaller organizations make effective use of general purpose development tools to strip headers and comments from programs and to transform these into documentation. More typical is the approach where supervisors, in some cases second level supervisors, assume the major responsibility for the review of documentation.

A very significant part of the overall activity in the simulation field is version control. New assignments frequently consist of assembling existing modules, some with minor changes, into a new configuration. File management systems can be used very effectively to assist in this process. Editors and preprocessors (usually without extensive analysis features) are other typical tools currently used in this environment.

By and large the scientific programming organizations have the technical ability to acquire and install software tools. They may lack specific information on tools suitable for their environment, the resources for the introduction, and frequently also the motivation to devote part of their effort to software engineering. Because of the recognized difficulties in documentation and maintenance, the second level supervisors will be particularly receptive to tools that can simplify the work in these areas.

SECTION 4

USER TOOL NEEDS

This section discusses those aspects of user tool needs that are pertinent to the development of guidance for the introduction of tools. Subsection 4.1 considers organizational factors of tool needs that are largely independent of the application area. This is followed in subsection 4.2 by a detailed identification of tool features desired in the target environments described in Section 3. Even experienced tool users can be faced with severe problems in the adoption of new tools, and the needs that arise in this connection are addressed in subsection 4.3. The final part of this section describes resources available to the potential tool user for selection of specific tools to meet the needs characterized in the earlier parts.

4.1 ORGANIZATIONAL FACTORS IN TOOL NEEDS

The objectives of tool usage (and hence the objectives of many tool features) are determined largely by the user's organizational environment and by the management level that authorizes tool acquisitions. Because these considerations hold for all application areas, they are discussed at the beginning of this section. For a tool to be readily accepted, it must help in areas of concern to the management that authorizes the acquisition and introduction activities. Thus, if management considers program documentation to be a particularly critical area it may be difficult to obtain authorization for the introduction of test tools. The organizational entities that may be involved in the acquisition and use of software tools are described under the headings of:

Software Development Organization,
Project Management, and
Functional Management.

4.1.1 Software Development Organization

The term development is used in a broad sense that includes all the activities directly involved in generating and maintaining programs. Practically all software development organizations desire tools that:

Increase productivity.
Reduce skill requirements.
Automate routine aspects of software design.
Help in software maintenance.

To some extent the last three items are individual facets of the first. At the present time there are few tools that are specifically aimed at a reduction of skill requirements. The creative and cognitive skills required for designing a

sound software structure are not easily packaged into a software tool. However, the automation of routine tasks is a very widely addressed tool objective. Because they relieve creative personnel from tedious aspects of design, coding, and testing, these tools compensate partly for the lack of those that reduce the skill levels. They also contribute to increased productivity. Examples of such tools are editors and precompilers used for the preparation or conversion of source files, formatters for the preparation of reports, and sort/merge programs. The most common tool features that automate routine tasks are editing, formatting, comparison, translating, and scanning. Tools that help in software maintenance include most of those cited for the automation of routine tasks plus file managers or library systems. In addition, maintenance may make use of special functions in editing or scanning tools, e. g., to locate variables or to strip code from a source file. The latter feature is useful for creating documentation from the program comments.

All of the tool functions and features enumerated here are of direct benefit to the software professional, and there is seldom any difficulty in introducing them at the working level if they provide a reasonably friendly user interface. Line management in the software development organization may need to be convinced that the cost of the tool acquisition and introduction will be recovered over a reasonably short time span. Note that the use of these tools is largely independent of emphasis on standards that may prevail in the using organization.

Where standards are in use, additional tools will be desired that either facilitate or enforce compliance. Among the former are program design language processors, and among the latter are code auditors. Environments that emphasize standards usually also demand discipline in the procedural aspects of software development such as version control, access to test cases, etc. File managers and library systems will be found helpful in enforcing this discipline. Tools that support standards will be readily accepted by a standards-minded line management. The professionals who have to use the tools may regard them with indifference or even hostility. Part of this is due to apprehension about having one's work scrutinized by "Big Brother", and part is due to obstacles to innovation (deviation from standards) which these tools may present. It is therefore important that tools of this type have a particularly good user interface so that potential complaints about their use can be minimized. Some tools, such as auditors, can be combined with the compiler so that they are automatically invoked when a new source file is submitted. This integration makes more efficient use of the computer and at the same time avoids problems at the user interface.

That they support or enforce standards is a particularly pertinent factor in connection with the introduction of software tools to Government agencies. Beyond the benefits that always attend uniformity of design practices. Government agencies will find it easier to interchange both programs and personnel if common standards can be adopted. Some of these benefits transcend the usual concerns of the software development organization. The broader aspects of tool usage to support software standards are discussed under Functional Management in 4.1.3.

Because Government agencies can have access to tools developed or in use by other Government organizations, they may particularly benefit from the appointment of a local toolsmith -- a person expert in tool usage who may be able to make software or minor hardware modifications that permit a tool to be used in a new environment. The role of the toolsmith was introduced several years ago as a specialist within a software development team in these words [BR0075]:

> (The team leader) needs a toolsmith, responsible for ensuring this adequacy of the basic service and for constructing, maintaining, and upgrading special tools -- mostly interactive computer services -- needed by his team. Each team will need its own toolsmith, regardless of the excellence and reliability of any centrally provided service, for his job is to see to the tools needed or wanted by his (team), without regard to any other team's needs. The tool-builder will often construct specialized utilities, catalogued procedures, and macro libraries.

The designation of a dedicated toolsmith within each team may be a higher degree of specialization than can be warranted in smaller software organizations. However, within each software environment that makes use of a single computing facility such a specialist will be found very effective and certainly very valuable for the introduction of new tools.

4.1.2 Project Management

Project management directs the software development on behalf of the ultimate user. It is usually more interested in the functional and interface aspects of the programs than in structural or standards aspects. Where project management is funding the acquisition of software tools, there may be heavy emphasis on tools that have an immediate payoff in terms of project objectives. Some of these tools may be software development tools but the nature of these is project dependent and can not be predicted.

There are, however, some software tools that make a direct contribution to project management, and this area of tools usage is expected to be expanded in the near future. Some programs of this kind are general purpose scheduling and reporting algorithms that share more of the characteristics of application programs than those of software tools. Others, however, are very specific to the software area and extract information from the software as it is being developed. These are appropriately described as software tools for project management and are further described below.

Software library systems have already been mentioned in the previous heading as tools that can support disciplined development and aid in software maintenance. For project management they can furnish the identification and date of the latest revision, current file size (number of statements), and change in size over a selected time interval. Either by themselves or in conjunction with the operating system log, these tools can also furnish reports on the total number of runs, the number of statement changes, the number of different test cases submitted, and the number of compilation failures or aborted runs. All of this information can be furnished in hard copy or interactively on a terminal. In

either format, tools furnish these data more conveniently and at a fraction of the cost of manual methods.

Tools for cost estimation are also important for the project management area. Development cost, life cycle cost, and computer cost aspects can be estimated by means of software tools. Development costs are estimated by automating an estimation algorithm such as [DOTY77]. Life cycle costs can be developed as an extension of the development costs, such as in the ESD model [JAME77], or from data on a system under development such as [PUTN78]. Computer cost aspects are estimated by sizing and timing tools. While the jury is still out on the accuracy of the cost estimates generated by these tools at present, there is little doubt that their use promotes systematic collection of software cost data and a methodical approach to software costing.

Since the emphasis in this report is on the introduction of tools to smaller programming environments, it should be noted that not all project management tools need to be very large systems. Management will frequently derive considerable benefits from small programs that automate follow-up on action items, receipt of deliverables from vendors, etc. Programs of this type can be applied in any environment regardless of size.

4.1.3 Functional Management

In organizations where software is being developed for more than one project, the individual development groups usually report to a common management level which is referred to here as the functional management or computing function management. Because personnel must be periodically reassigned to new projects, functional management will usually be interested in uniformity of practices among projects so that retraining can be minimized. Computing function management can thus be expected to be standards-oriented and to support the introduction of tools that enforce standards. This management level is usually inclined to take a long range view and may favor the acquisition of tools that primarily benefit later software lifecycle phases, e. g., requirements analyzers (although these are used during the definition stage, the major benefits are usually reduced maintenance costs during the operation phase).

Functional management is usually also involved in another important aspect of the introduction of software tools: it must furnish or allocate the facilities for the execution of the tools. Very few computing facilities have excess capacity, and this is particularly true for Government computing facilities. Therefore, the management of the computing function may object to the introduction of tools that extend the execution time or that add job steps to frequently run programs. Where a tool has a significant adverse impact on throughput, the benefits of that tool in areas of concern to functional management should be highlighted: increased programmer productivity, adherence to standards, or improved software quality.

Because tool integration avoids repeated reformatting and multiple data retrievals, it reduces computer usage and supports the goals of functional management. It is at this management level that the greatest recognition of the benefits of tool integration efforts can be expected.

Functional management will also be interested in tools useful for the management of the computing facility, e. g., those that allocate charges to users. that report on the operation of the current facilities, and simulation tools that aid in planning of improvements. The features of instrumentation, resource utilization, simulation, and statistical analysis support the capabilities of such tools.

4.2 APPLICATION FACTORS IN TOOL NEEDS

The following discussion focuses on the needs of the two environments that were identified in Section 3 as the primary targets for the introduction of software tools: the smaller MIS organization and the smaller scientific programming organization. In the studies leading to the definition of tool needs, six application areas were considered: business-oriented batch systems. management information systems, office automation systems, online transaction driven systems, real time command and control systems, and scientific or engineering programs. It was found that the tool needs of the first four of these were very similar, and this entire group is encompassed by the discussion in 4.2.1 below. Also, the software tool needs of the last two categories were identical, and these are described in 4.2.2 below. Within this subsection it is assumed that the tool types and features required for the general software development organization [4.1.1 above] are provided, and therefore only the supplements dictated by the specific application areas are discussed.

4.2.1 Tool Needs of Smaller MIS Organizations

One of the distinctive tool needs of this environment arises from the use of COBOL and the inefficiencies of COBOL compilers that were mentioned earlier [3.3]. A sizeable number of commercial tools have been developed to improve the performance of COBOL programs. Two specific techniques have been found particularly helpful in this area: modifying the object code for improved performance (the significant tool feature for this is optimization), and determining and simplifying the parts of the program which account for the bulk of the run time (the significant tool feature for this is tuning). Of course both of these can also be used together. Tuning is part of the dynamic analysis function. It generally requires instrumenting the program, I. e., the insertion of code that counts the number of accesses to the program segments of interest. Once this is done, other attributes of the program's structure and performance can also be evaluated, and such options are provided in several of the optimization tools. In connection with the introduction of tools into the smaller MIS organization, it is suggested to avoid such additional capabilities in the tool initially because they extend the run-time of the instrumented program, and they make the user interface more complex than it needs to be.

Tools can be used very effectively to aid in program conversion (e. g.. when a new computer is being installed) which can present many problems in the smaller MIS environment. Over 270 conversion tools are listed in a publication of the Federal Conversion Support Center [FCSC80]. The listing includes tools that facilitate conversion (e. g., translators), as well as programs that may eliminate the need for conversion (e. g., emulators).

Because of the heavy involvement of the MIS applications area in the manipulation of data structures, tools that simplify data base updating and restructuring are another specific need of this environment. While program libraries and general purpose file management systems mentioned in 4.1.1 can be of some help for updating, specialized systems for data base management are preferred. A number of these are commercially available, and they frequently combine access control, archiving (or providing an audit trail), auditing for completeness and reasonableness of the inserted data, and restructuring with the update capability. Data encryption is offered as an optional feature of some tools but is not considered essential for the smaller MIS environment.

4.2.2 Tool Needs of the Smaller Scientific Programming Environment

Just as the use of COBOL is responsible for some specific tool needs in the MIS environment, the use of FORTRAN, the current leading language in the scientific programming environment, has in the past also been a strong motivator for specific tools, in this case pre-processors for structured languages. Pre-processors frequently represent the most advanced software tool in the inventory of smaller scientific programming organizations. Even though pre-processors will continue to be used, it is suggested that a forward looking program for the introduction of tools to the smaller scientific programming environment not emphasize this tool type unduly. One of the reasons is that, in scientific programming for the defense-connected sector, FORTRAN is being replaced by languages which inherently support structured programming practices, and another reason is that suitable control constructs are evolving for FORTRAN [ANSI78]. But more fundamental is the need to educate the scientific programmer in the smaller organization to the benefits of a methodical approach to program development of which structured programming is but a part.

This need can be met by a general purpose development tool package that takes the drudgery out of some of the routine programming steps. One such package, described in the professional literature [KERN76], is in the public domain. This approach, which is cited only as an example, involves a number of independent utilities that can be invoked individually or interactively by means of a command line processor to do editing, file management, formatting, and pre-processing. The efficient application of the tools poses an intellectual challenge that may be particularly motivating for scientific and engineering personnel who are the programmers in this environment. Software tool packages patterned along these lines are in use in some smaller scientific environments, and the user reaction seems to be favorable.

Once a scientific programming organization is committed to a disciplined development approach (and as previously explained some of the smaller organizations are already at that point), needs for many static analysis features may arise, including code auditing, completeness checking, consistency checking, error checking, and statistical analysis. Tools with these capabilities will be particularly desired for design analysis programs supporting critical applications (nuclear industry, aircraft structures) and for simulations which furnish output to physically active equipment (e. g., moving base flight simulators). At present, smaller organizations may find it difficult or impossible to acquire these tools in a useful format. Once a general purpose development tools package is in use, the additional checking tools can be developed in-house. Commercial sources may become interested in adding checking functions to an established basic tools package.

Real-time command and control systems pose additional requirements that may require dynamic analysis features. At present, this applications area is primarily served by large organizations that make extensive use of tools, the

4.2.3 Summary of Tool Features Determined by the Application

Table 4-1 lists common and application-dependent tool features. The first column lists features desired by all software development organizations as described in 4.1.1. The features shown in the table are the ones most desired by new tool users. The selection involved some judgment regarding the priorities that exist within the software development organization and its user community. Thus, that error checking is found in the scientific column but not in the MIS column does not imply that this feature is not suitable or not important in the MIS environment. It does mean that most smaller MIS developers will place a lower priority on error checking than on the features listed in the MIS column.

TABLE 4 - 1 FEATURES DETERMINED BY THE APPLICATION

	Features Needed For	
Common	MIS Programming	Scientific Programming
Editing	Optimization	Auditing (Code)
Scanning	Tuning	Completeness Checking
Formatting	Restructuring	Statistical Analysis
Comparison	Auditing (Data)	Error Checking
Translation		Consistency Checking
Management		

4.3 NEEDS OF OTHER ENVIRONMENTS

Although large and very large software organizations are in most cases already users of general purpose software tools, they may benefit from programs aimed at improving access to tools, standardization of tool interfaces, or establishing minimum requirements for tool documentation and diagnostics. Because these organizations (which will be collectively called "larger") have multiple general purpose tools in inventory, the integration of tools is particularly pertinent for them. There are at present no clear indications of the direction that tool integration should take. However, there is a considerable effort being devoted to the area of programming environments within NBS and elsewhere, and tool integration is an important aspect of these activities [BRAN81, IEEE81].

The integration of tools provides two primary benefits: a simplified user interface and reduced utilization of computer resources. The simplified user interface is achieved by requiring less file manipulation for converting the output of one tool into an input for another, by consistent tool calling conventions, input commands, and output formats, and by the capability for invoking processing by several tools with a single command string. These benefits will in turn simplify documentation and training and in general improve the user acceptance of a system of multiple tools.

The reduced utilization of computer resources is partly due to the factors just enumerated, particularly the avoidance of file manipulations, and partly due to the possibility of combining computer-intensive operations such as parsing and searching which are now carried out separately. As mentioned in 4.1.3, the managers of the computing function will particularly appreciate these latter benefits. The reduction in running time and storage requirements together with the benefits due to the simplified user interface promise a high payoff for efforts in the tool integration area.

A significant step for the integration of tools developed by different sources is represented by the NBS/ICST study of compiler-based tools [BRAY81]. The association of the tool with the compiler provides access to at least two (source and object) and sometimes three (parsed code) representations of the program, and also makes the data and structure checking features of the compiler available to the tool. A possible disadvantage of this approach is that a compiler pass may be required in order to invoke a tool. The adherence to a single (or at least compatible) file format for multiple tools can be readily enforced by compiler basing.

The integration of tools is very significant for extending tool usage in environments where general purpose tools are already in use. It is of lesser importance for the introduction of tools to environments that had no prior experience with general purpose software tools, and it is therefore addressed here in only a limited way.

A topic partly contained within the area of tool integration is the standardization of tool commands and output formats. The lack of

standardization is particularly obvious and disadvantageous in editors and related tools (including word processors). These are among the most widely used tools, they are frequently the medium through which the input to other tools is processed, and there is better agreement than in most other areas on the functions which the tool is required to furnish. There is thus ample motivation to standardize but very few concrete accomplishments.

There are at present many different methods for cursor positioning, different commands for deleting characters, words, or lines, and different procedures as well as commands for string search or substitution. These inconsistencies cause errors, necessitate multiple training periods, and certainly constitute a deterrent to tool usage. In view of the basic need for an editor in the use of many tools, the lack of standardization of editor commands must be regarded as an obstacle to the introduction of tools.

4.4 RESOURCES FOR TOOL SELECTION

In subsections 4.1 and 4.2 a number of generic tool types and features have been identified as suitable for the introduction of tools to specified organization and application environments. The present subsection discusses the additional steps necessary for the actual selection of a tool.

Catalogues of software tools are available from the commercial tool developers, computer manufacturers, and computer users' groups. For obvious reasons, the offerings in each of these are restricted although the restriction imposed by the last of these may be an appropriate one if only the computer type addressed by that users' group is available as a tool host and if the group has conducted a comprehensive survey of suitable tools.

A recent publication by the National Bureau of Standards is particularly helpful for the introduction of tools to smaller programming environments [HOUG82]. It contains cross references by tool classification (function) and features which makes it especially suitable for use with the tool identifications used in this report.

Once a tool that meets the functional requirements is identified, the main section of the catalogue must be consulted to see whether the tool is usable on an available computer, whether it handles an appropriate source language (or other input format), and whether it can be obtained in a suitable implementation language. Other considerations are the licensing arrangements, availability of documentation and training, and the computer resource utilization.

Some difficulties are usually encountered that must be resolved by language conversions (of either input or the tool) or other tool modifications. Consultation with a toolsmith will be valuable in this connection. Government agencies will want to know whether other agencies are currently using the tool, and a central tool usage catalogue will be a beneficial facility. Access at a remote computer can be a very effective first step in a detailed evaluation of the tool. Hosting problems may be overcome by remote access even on a longer term basis.

SECTION 5

DEVELOPMENT OF EVENT SEQUENCES

While the preceding sections have discussed the tool needs in selected environments, this section describes the detailed events that will lead to successful use of tools. The first subsection describes the purpose and rationale for an event sequence. and the second subsection recommends a specific event sequence for the smaller MIS and scientific environments.

5.1 PURPOSE OF EVENT SEQUENCES

The management of any significant project requires that the work be divided into tasks for which completion criteria can be defined. The transition from one task to another is called an event, and to permit orderly progress of the activities, here the introduction of a software tool, the scheduling of these events must be determined in advance. A general outline for such a schedule is provided by the event sequence described in the next subsection. The actual calendar time schedule will depend on many factors which must be determined for each specific tool use (particularly on the time required for procurement of the tool and training). One of the formats used for the event sequence is consistent with the Critical Path Method (CPM) of project scheduling and can be used with that technique for the development of an optimum calendar time schedule.

Most of the activities included in the event sequence are obviously necessary but a few were included specifically to avoid difficulties encountered in previous tool procurements. Quite frequently tools were obtained 'through the side door' without adequate consideration of the resources required for the effective employment of the tool and without determination by a responsible manager that the tool served a primary need of the organization. Tools acquired in this manner were seldom used in an optimal way and were sometimes discarded. Experiences of this type are not conducive to gaining widespread acceptance of tools in the smaller programming environments where the activities required for the introduction of tools will, under the best of circumstances, impose a severe drain on resources. A key feature of the proposed approach is, therefore, that tool usage will be initiated only in response to an expressed management goal for software development or for the entire computing function.

Difficulties in the introduction of tools can arise in three areas:

Organizational obstacles
Problems arising from the tools
Obstacles in the computer environment

The individual activities described below as well as the ordering of the event sequence are designed to eliminate as many of these difficulties as possible. They are most effective with regard to the first category and probably least effective with regard to the last category. The need for involving a reponsible

management level in the tool introduction has already been mentioned, and this is indeed the key provision for avoiding organizational obstacles. "Responsible management" is that level that has the authority to obligate the resources required for the introduction process. The scope of the resource requirement will become clearer after all introduction activities have been described. Because the criterion for the selection of the management focus is its ability to commit funds, this management level is hereafter referred to as funding management. In some organizations this may be the project management as defined in 4.1.2, in some it may be functional management as defined in 4.1.3, and in yet others it may be an agency or department management not specifically identified with a computing function. It should be involved in at least the following activities associated with the introduction of tools:

1. Identifying the goals to be met by the tool (or by the technique supported by the tool), and assigning responsibility for the activities required to meet these goals.

2. Approving a detailed tool acquisition plan that defines the resource requirements for procurement and in-house activities.

3. Approving the procurement of tools and training if this is not explicit in the approval of the acquisition plan.

4. Determining after some period of tool use whether the goals have been met.

Additional organizational obstacles must be overcome by actions of the software management (local management of the organization that will introduce the tool). A pitfall that must be avoided is assigning the details of the tool acquisition as a sideline to an individual who carries many other responsbilities. Even in a small software organization (up to 14 programmers), it should be possible to make the tool introduction the principal assignment of an experienced individual with adequate professional background. This person is referred to as the software engineer. In medium size organizations (15 to 39 programmers) several individuals may be involved in software engineering tasks (not restricted to tool usage), and this may constitute a software engineering function.

Further, the event sequence includes activities of a toolsmith who will not be the same person as the software engineer in most cases. The former assignment requires expertise in systems programming and specialized knowledge of the tool to be introduced. The duties of the software engineer involve planning project management, and obtaining cooperation from a variety of individuals and organizations. Where there is a software engineering function, the toolsmith is typically a member of it.

Obstacles arising from the tools themselves are expected to be avoided in the event sequence by a careful, methodical selection of tools. In particular, distinct contributions to the tool selection are specified for software management and the software engineer. Software management is assigned responsibility for:

Identifying tool objectives.

Approving the acquisition plan (it may also require approval by funding management).

Defining selection criteria.

Making the final selection of the tool or the source.

The software engineer is responsible for:

Identifying candidate tools.

Applying the selection criteria (in informal procurement) or preparing RFP inputs (in formal procurement).

Preparing a ranked list of tools or sources.

Further, the ultimate user of the tool is involved in the recommended event sequence in reviewing either the list of candidate tools or, for formal procurement, the tool requirements.

This distribution of responsibilities reduces the chances of selecting a tool that (1) does not meet the recognized needs of the organization, (2) is difficult to use. (3) requires excessive computer resources, or (4) lacks adequate documentation. The repeated exchange of information required by the process outlined above will also avoid undue emphasis on very short-term objectives which may lead to selection of a tool on the basis of availability rather than suitability.

The obstacles to tool usage that reside in the computer environment are primarily due to the great diversity of computer architectures and operating system procedures, and to the lack of portability in most software tools. Activities associated with the introduction of tools can only modestly alleviate these difficulties. The event sequence provides the following help in this area:

1. A methodical process of identifying candidate tools and selecting among these on the basis of established criteria. This will avoid some of the worst pitfalls associated with "borrowing" a tool from an acquaintance or procuring one from the most accessible or persuasive tool vendor.

2. The assignment and training of a toolsmith who can make minor modifications to both the computer environment and the tool. This is expected to provide relief where there are version-related or release-related incompatibilities with the operating system, or where the memory requirements of the tool exceed the capabilities of the installation. In the latter case, remedies may be provided by removing tool options or by structuring the tool program into overlays.

The event sequence described below is conceived as a procedure generally applicable to the introduction of tools to Federal agencies falling into pertinent programming environment categories. For this reason, a systematic reporting of the experience with the introduction process as well as with the tool is desirable. The evaluation plan and the evaluation report specified in the event sequence support these goals.

5.2 RECOMMENDED EVENT SEQUENCE

The event sequence described in this subsection is applicable to both the smaller MIS and scientific programming environments. The general scope of the introduction activities and their sequence are identical for the two environments. Because of differences in tool requirements, personnel qualifications, and organizational structure, some differences in the content of the individual events will be expected. The event sequence addresses only the introduction of existing tools. Where a newly developed tool is introduced, a considerable modification of the activities and their sequence will be necessary.

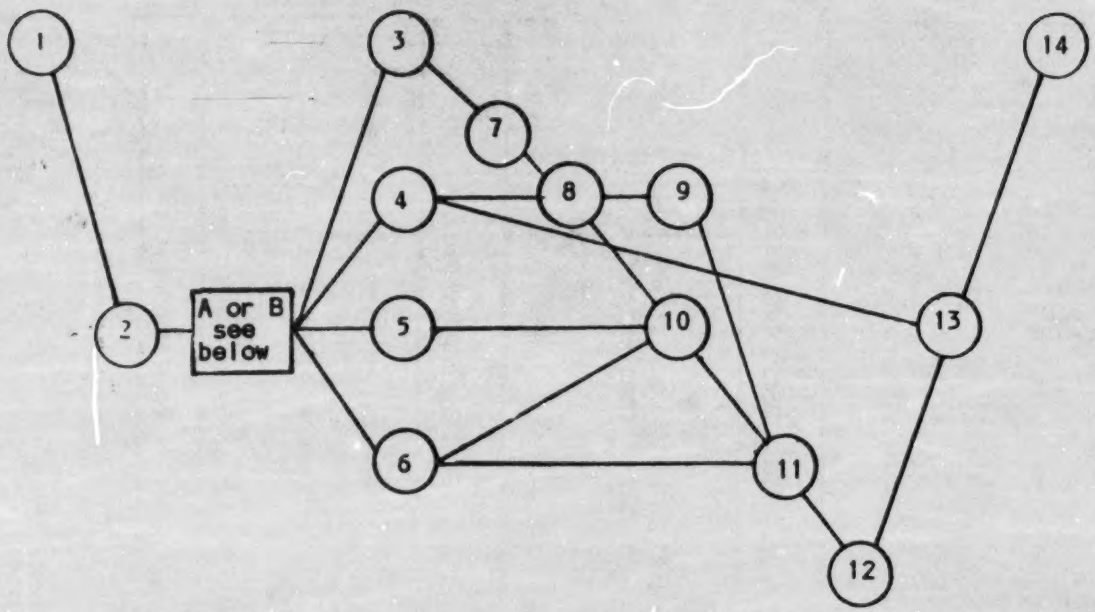
The recommended event sequence allows for two procurement methods: informal procurement (e. g., by purchase order) or formal procurement by request for bids. Obviously, the latter is much more time consuming but it may lead to the procurement of better or cheaper tools. Acquisition of tools from the General Services Administration or from other Government agencies should follow the informal procurement steps even when there is no procedural requirement for this. As mentioned above, tool acquisitions which do not obtain the concurrence of all affected operational elements frequently do not achieve their objectives.

The presentation of the event sequence in Table 5-1 is tailored to tools which are being introduced for the first time into a user community which shares software support information (e. g., a Federal agency or a private sector company). As a result, some steps are shown which can be combined or eliminated where less formal control is exercised or where plans or modifications required for the introduction of a tool are available from a prior user. The event sequence is intended to cover a wide range of applications, and it was constructed with the thought that it is easier for the tool user to eliminate steps than to be confronted with the need for adding some that had not been covered in this volume.

The key functions which contribute to the introduction of tools are listed across the top of Table 5-1, and events for which each function is responsible are listed in the column under it. The preferred order of tasks for each function can thus be directly found from this table. The precedence relationships between events is shown in graph form in Figure 5-1. This figure will be found particularly helpful for scheduling activities by the Critical Path Method and for the general development of a project schedule. The numbering of events is the same in Table 5-1 and Figure 5-1. A detailed description of each of the numbered events, and of the activities associated with it, is presented following the table and figure.

TABLE 5 - 1 EVENT SEQUENCE FOR TOOL INTRODUCTION

FUNDING MANAGEMENT	SOFTWARE MANAGEMENT	SOFTWARE ENGINEER	TOOL USER
1. Goals	2. Tool Objectives		
<---------- Acquisition, see A or B below ---------->			
3. Procure tool	A <----------	4. Evaluation plan	
	A <----------	5. Toolsmithing plan-S	
	A <----------	6. Training plan ------	participates
7. Receive tool		8. Acceptance test	
	9. Orientation	10. Modifications-S	
		<-------- 11. Training --------->	
			12. Use
A <----------	A <----------	13. Evaluation report	
14. Goals met?			
A. Acquisition Activities for Informal Procurement			
A <--------	A1. Acquisition plan A2. Select'n criteria	A3. Ident. candidates A5. Score candidates	A4. Review
	A6. Select tool		
	continue with step 3 above.		
B. Acquisition Activities for Formal Procurement			
A <--------	B1. Acquisition plan	B2. Technical req'mts	B3. Review
A <----------	A <----------	B4. Generate RFP	
B5. Issue RFP		B6. Proposal Evaluation	
	B7. Select source		
	continue with step 3 above.		
A = Approval required		S = Toolsmith responsibility	



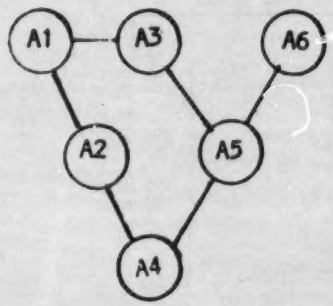
EVENTS

1. Goals
2. Tool objectives
3. Procure tool
4. Evaluation plan
5. Toolsmithing plan
6. Training plan
7. Receive tool
8. Acceptance test
9. Orientation
10. Modifciations
11. Training
12. Use
13. Evaluation Report
14. Goals met?

A1. Acquisition plan
A2. Selection criteria
A3. Identify candidates
A4. User review
A5. Score candidates
A6. Select tool

B1. Acquisition plan
B2. Technical req'mts
B3. User review
B4. Generate RFP
B5. Issue RFP
B6. Proposal evaluation
B7. Select source

A. INFORMAL PROCUREMENT



B. FORMAL PROCUREMENT

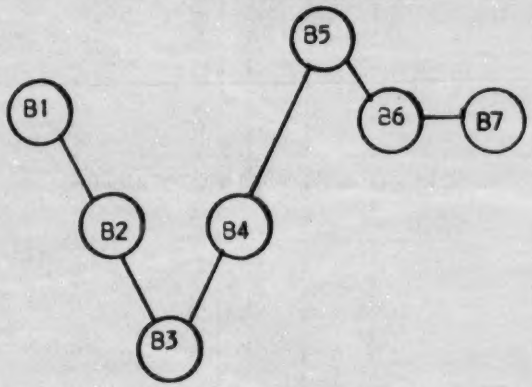


FIGURE 5 - 1 PRECEDENCE RELATION FOR EVENT SEQUENCE

1. Goals

The goals to be accomplished should be identified in a format that permits later determination (event 14) that they have been met. Typical goal statements are: reduce average processing time of COBOL programs by one-fifth; achieve complete interchangeability of programs or data sets with organization Y; adhere to an established standard for documentation format.

The statement of goals shall also identify responsibilities, in particular the role that headquarters staff organization may have and coordination requirements with other organizations. Where a decentralized management method is employed, the statement of goals may have associated with it a not-to-exceed budget and a desired completion date. Once these constraints are specified, funding management may delegate the approval of the acquisition plan to a lower level.

2. Tool Objectives

The goals generated in event 1 are translated into desired tool features (e.g., see Table 4-1), and requirements arising from the development and operating environment are identified. Constraints on tool cost and availability may also be added at this event. A typical statement of tool objectives for a program formatter is: Provide header identification, uniform indentation, and the facility of printing listing and comments separately for all FORTRAN X3.9-1978 and ABC Extended FORTRAN programs. Program must run on our ABC computer under XOSnn. Only tools which have been in commercial use for at least 1 year and at no less than N different sites shall be considered.

At this point the sequence continues with either A1 or B1 below.

- - - - -

A. Acquisition Activities for Informal Procurement

A1. Acquisition Plan

The acquisition plan communicates the actions of software management both upward and downward. The plan may also be combined with the statement of the tool objectives (event 2). The acquisition plan should include the budgets and schedules for subsequent steps in the tool introduction, a justification of resource requirements in the light of expected benefits, contributions to the introduction expected from other organizations (e. g., the tool itself, modification patches, or training materials), and the assignment of responsibility for subsequent events within the software organization, particularly the identification of the software engineer. Minimum tool documentation requirements shall also be specified in the plan.

A2. Selection Criteria

The criteria shall include a ranked or weighted listing of attributes that will support effective utilization of the tool by the user. Typical selection criteria are:

Accomplishment of specified tool objectives.
Ease of use.
Ease of installation.
Minimum processing time.
Compatibility with other tools.
Low purchase or lease cost.

Most of these criteria need to be factored further to permit objective evaluation, but this step may be left up to the individual who does the scoring. Together with the criteria (most of which will normally be capable of a scalar evaluation), constraints which have been imposed by the preceding events or are generated at this step should be summarized.

A3. Identify Candidate Tools

This is the first event for which the software engineer is responsible. The starting point for preparing a listing of candidate tools is a comprehensive tool catalogue, such as [HOUG82]. A desirable but not mandatory practice is to prepare two lists, the first of which does not consider the constraints and contains all tools meeting the functional requirements. The Cross-Reference by tool features in the appendices of [HOUG82] will be found particularly valuable in generating this list of candidates. For the example used in event 2, a program formatting tool, 16 entries are found there. Some of these may be eliminated by further review of their description in the body of the catalogue (e. g., because they don't process the specified FORTRAN dialects). For the remaining viable candidates, literature should be requested from the developer, and this is examined for conformance with the given constraints. At this point a second list is generated, containing tools that meet both the functional requirements and the constraints. If this list does not have an adequate number of entries, relaxation of some constraints will have to be considered.

A4. User Review of Candidates

The user reviews the list of candidate tools prepared by the software engineer. Because few users can be expected to be very knowledgeable in the software tools area, specific questions may need to be raised by software management such as: "Will this tool handle the present file format? Are tool commands consistent with those of the editor? How much training will be required?" Adequate time should be budgeted for this review and a due date for responses should be indicated. Because the user views this as a far-term task, of lower priority than many immediate obligations, considerable follow-up by line management will be required. If tools can be obtained for trial use, or if a demonstration at another facility can be arranged, it will make this step much more significant.

A5. Score Candidates

For each of the criteria previously identified a numerical score is generated on the basis of information obtained from vendor's literature, from demonstration of the tool, from the user's review, from observation in a working environment, or from comments of prior users. If weighting factors for the criteria are specified, the score for each criterion is multiplied by the

appropriate factor and the sum of the products represents the overall tool score. Where only a ranking of the criteria was provided, the outcome of the scoring may be simply a ranking of each candidate under each of the criteria headings. Frequently a single tool is recognized as clearly superior in this process.

A6. Select Tool

This decision is reserved for software management in order to provide review of the scoring, and also to permit additional factors which were not expressed in the criteria to be taken into consideration. For example, a report might just have been received from another agency that the selected vendor did not provide adequate service. If the selected tool was not scored highest, the software engineer should have an opportunity to review the tool characteristics thoroughly to avoid unexpected installation difficulties. The selection concludes the separate sequence for informal procurement. Continue with event 3.

- - - - -

B. Acquisition Activities for Formal Procurement

B1. Acquisition Plan

The plan generated here must include all elements mentioned under A1 plus the constraints on the procurement process (e. g., set-aside for high labor surplus areas) and the detailed responsibilities for all procurement documents (statement of work, technical and administrative provisions in the Request for Proposal, etc.).

B2. Technical Requirements Document

The technical requirements document is an informal description of the tool requirements and the constraints under which the tool has to operate. It will utilize much of the material from the acquisition plan but should add enough detail to support a meaningful review by the tool user.

B3. User Review of Requirements

The user reviews the technical requirements for the proposed procurement. As in the case of event A4, the user may need to be prompted with pertinent questions, and there should be close management follow up in order to get a timely response.

B4. RFP Generation

From the technical requirements document and the user comments on it, the technical portions of the RFP can be generated. Usually these include:

1. A specification of the tool as delivered. This should include applicable documents, a definition of the operating environment, and the quality assurance provisions.

2. A statement of work governing the tool procurement. This should state any applicable standards for the process by which the tool is generated (e. g., configuration management of the tool), and documentation or test reports to be furnished with the tool. Training and operational support requirements are also identified in the statement of work.

3. Proposal evaluation criteria and format requirements. Evaluation criteria are listed in the approximate order of importance. Subfactors for each may be identified. Restrictions on proposal format (major headings, page count, desired sample outputs) may also be included.

B5. Solicitation of Proposals

This activity is carried out by administrative personnel. Capability lists of potential sources are maintained by most purchasing organizations. Where the software organization knows of potential bidders, their names should be made known to the procurement office. When responses are received, they are screened for compliance with major legal provisions of the RFP.

B6. Technical Evaluation

Each of the proposals received in response to the RFP is evaluated against the criteria previously established. Failure to meet major technical requirements can lead to outright disqualification of a proposal. Those deemed to be in "the competitive range" will be assigned point scores that will then be used together with cost and schedule factors that are being separately evaluated by administrative personnel.

B7. Source Selection

On the basis of the combined cost, schedule, and technical factors, a source for the tool is selected. If this was not the highest rated technical proposal, prudent management will require additional reviews by software management and the software engineer to determine that it is indeed acceptable.

The source selection concludes the separate sequence for formal procurement. Continue with event 3.

- - - - -

3. Procure Tool

In addition to determining that the cost of the selected tool is within the approved budget, the procurement process will also consider the adequacy of licensing and other contractual provisions and compliance with the "fine print" associated with all Government procurements. The vendor's responsibility for furnishing the source program, for meeting specific test and performance requirements, and for tool maintenance need to be identified. In informal procurement, a period of trial use may be considered if this had not already taken place under one of the previous events.

If the acquisition plan indicates the need for outside training, the ability of the vendor to supply the training and the cost advantages from combined procurement of tool and training should be investigated. If substantial savings can be realized through simultaneous purchase of tool and training, procurement may be held up until outside training requirements are defined (event 6).

4. Evaluation Plan

The evaluation plan is based on the goals identified in event 1 and the tool obectives derived from these in event 2. It describes how the attainment of these objectives is to be evaluated for the specific tool selected. Typical items to be covered in the plan are milestones for installation, dates and performance levels for the initial operational capability and for subsequent enhancements. Where improvements in throughput, response time, or turn-around time are expected, the reports from which these data are to be obtained should be identified. Responsibility for tests, reports and other actions should be assigned in the plan. A topical outline of the Evaluation Report should be included.

The procedure for the acceptance test is a part of the Evaluation Plan, although in a major tool procurement it may be a separate document. It lists the detailed steps necessary to test the tool in accordance with the procurement provisions when it is received, to evaluate the interaction of the tool with the computer environment (e. g., adverse effects on throughput), and for generating an acceptance report.

5. Toolsmithing Plan

The plan will describe the selection of the toolsmith, the responsibilities for the adaptation of the tool, and the training which will be required. The toolsmith should preferably be an experienced system programmer, familiar with the current operating system. Training in the operation and installation of the selected tool in the form of review of documentation, visits to current users of the tool, or training by the vendor must be arranged. The toolsmithing plan is listed here as an event for which the software engineer is responsible, and in the discussion of further events it is assumed that the toolsmith will work under the direction of the software engineer. The toolsmithing plan should be approved by software management.

6. Training Plan

The training plan should first consider the training inherently provided with the tool, e. g., documentation, test cases, on-line diagnostics, HELP. These features may be supplemented by standard training aids supplied by the vendor for in-house training such as audio or video cassettes and lecturers. Because of the expense, training sessions at other locations should be considered only where none of the previous categories is available. The number of personnel to receive formal training should also be specified in the plan, and adequacy of in-house facilities (number of terminals, computer time, etc.) should be addressed. If training by the tool vendor is desired, this should be identified as early as possible to take permit training to be procured with the tool (see

step 3). User involvement in the preparation of the training plan is highly desirable, and coordination with the user is considered essential. The training plan is normally prepared by the software engineer and approved by software management. Portions of the plan should be furnished to procurement staff if outside personnel or facilities are to be utilized.

7. Tool Received

The tool is turned over by the procuring organization to the software engineer.

8. Acceptance Test

The software engineer or staff test the tool. This is done as much as possible in an "as received" condition with only those modifications made that are essential for bringing it up on the host computer. A report on the test is issued. After approval by software management it constitutes the official acceptance of the tool.

9. Orientation

When it has been determined that the tool has been received in a satisfactory condition, software management holds an orientation meeting for all personnel involved in the use of the tool and tool products (reports or listings generated by the tool). The main purpose is to communicate as directly as possible the objectives of the tool use, such as increased throughput or improved legibility of listings. Highlights of the evaluation plan should also be presented, and any changes in duties associated with the introduction of the tool should be described. Personnel should be reassured that allowance will be made for problems encountered during the introduction, and that the full benefits of the tool may not make themselves felt for some time.

10. Modifications

This step is carried out by the toolsmith in accordance with the approved toolsmithing plan. It includes modifications of the tool itself, of the documentation, and of the operating system. In rare cases some modification of the computer proper may also be necessary (channel assignments, etc.). Typical tool modifications involve deletion of unused options, changes in prompts or diagnostics, and other adaptations made for efficient use in the prevailing environment. Documentation of the modifications is an essential part of this event.

Vendor literature for the tool is reviewed in detail and is tailored for the prevailing computer environment and for the tool modifications which have been made. Deleting sections which are not applicable can be just as useful as adding material that is required for the specific programming environment. Unused options shall be clearly marked or removed from the manuals. If there is some resident software for which the tool should not be used (e. g., because of language incompatibility or conflicts in the operating system interface), warning notices should be inserted into the tool manual.

11. Training

Training is a joint responsibility of the software engineer and the tool user. The former is responsible for the content (in accordance with the approved training plan), and the latter should have control over the length and scheduling of sessions. Training is an excellent opportunity to motivate the user to utilize the tool. The tool user should have the privilege of terminating steps in the training that are not helpful and of extending portions that are helpful but in which greater depth is desired. Training is not a one-time activity. Retraining or training in the use of additional options after the introductory period is desirable. This also provides an opportunity for users to talk about problems with the tool.

12. Use in the Operating Environment

The first use of the tool in the operational environment should involve the most qualified user personnel and minimal use of options. The first use should not be on a project with tight schedule constraints. Any difficulties resulting from this use must be resolved before expanded service is initiated. If the first use is successful, then use by additional personnel and use of further options may commence.

User comments on training, first use of the tool, and use of extended capabilities are prepared and furnished to the software engineer. Desired improvements in the user interface, speed or format of response, and in utilization of computer resources are appropriate topics. Formal comments may be solicited shortly after the initial use, after 6 months, and again after 1 year.

13. Evaluation Report

The software engineer prepares the Evaluation Report, using the outline generated in event 4. The user comments and observations of the toolsmith form important inputs to this document. Most of all, it must discuss how the general goals and the tool objectives were met. The report may include, of course, observations on the installation and use of the tool, cooperation received from the vendor in installation or training, and any other "lessons learned". Tool and host computer modifications shall be described in the report. It may contain a section of comments useful to future users of the tool. The report is approved by software management and preferably also by funding management.

14. Determine if Goals Are Met

Funding management receives the Evaluation Report and determines whether the goals established in event 1 have been met. This determination shall be in writing and it shall include:

Attainment of technical objectives.
Adherence to budget and other resource constraints.
Timeliness of the effort.
Cooperation from other agencies.
Recommendations for future tool acquisitions.

REFERENCES

ANSI78 American National Standards Institute. "Programming Language FORTRAN - American National Standard X3.9-1978"

BRAN81 Martha A. Branstad and W. Richards Adrion, "NBS Programming Workshop Report", NBS Special Publication 500-78. National Bureau of Standards, June 1981

BRAY81 Gary Bray et al., "Compiler-Based Programming Support Capabilities", NBSIR 81-2423, January 1982

BROO75 F. P. Brooks, Jr., The Mythical Man-Month, Addison-Wesley, Reading MA, 1975

DOTY77 D. Doty et al., "Software Cost Estimation Study" (2 vols), RADC-TR-77-220. Rome Air Development Center, August 1977

FCSC80 Federal Conversion Support Center, "Conversion Products/Aids Survey", Report No. GSA/FCSC-80-01 (138 pp.)

HECH81 H. Hecht, "A Survey of Software Tools Usage", NBS Special Publication, in preparation

HOUG81 R. C. Houghton, Jr., "Features of Software Development Tools", National Bureau of Standards, NBS Special Publication 500-74, February 1981

HOUG82 R. C. Houghton, Jr., "Software Development Tools", National Bureau of Standards, NBS Special Publication 500-88, March 1982

IEEE81 IEEE Computer Society, Computer, Special Issue on Programming Environments, April 1981

JAME77 T. G. James, Jr., "Software Cost Estimating Methodology", NAECON'77 Proceedings, pp. 22-28, Dayton OH, May 1977

KERN76 Brian Kernighan and P. J. Plauger. Software Tools, Addison-Wesley, Reading MA, 1976

PUTN78 Lawrence H. Putnam, "Example of an Early Sizing, Cost and Schedule Estimate for an Application Software System", Proc. COMPSAC'78, pp. 827-832, November 1978

REIF80 D. J. Reifer and H. A. Montgomery, "Software Tool Taxonomy", Report SMC-TR-004, Software Management Consultants, Torrance CA, June 1980

APPENDIX

WORKSHOP ON PHASING OF SOFTWARE TOOLS

NATIONAL BUREAU OF STANDARDS
Lecture Room B, Administration Building
Monday, 18 May 1981

AGENDA

0900 - 0930	Welcome to NBS	M. Branstad, NBS
0930 - 1000	Software Automation Project and Objectives of the Workshop	R. C. Houghton, NBS
1000 - 1015	Coffee Break	
1015 - 1100	Survey of Software Tool Usage	H. Hecht, SoHaR
1100 - 1215	Tools Introduction Experience	
	NASA Langley	S. Voigt
	Naval Air Development Center	H. Stuebing
	NASA Goddard	F. McGarry
1215 - 1313	Lunch	
1315 - 1400	Guidelines for Phasing Software Tools Into Development Environments	H. Hecht, SoHaR
1400 - 1445	Discussion Groups	
1445 - 1500	Coffee Break	
1500 - 1545	Event Sequence for Tool Introduction	H. Hecht, SoHaR
1545 - 1630	Discussion Groups	
1630 - 1700	Wrap-Up	

WORKSHOP ON PHASING OF SOFTWARE TOOLS

PARTICIPANTS

Leo Beltracchi
Nuclear Regulatory Commission

Martha Branstad
National Bureau of Standards

Arthur F. Chantker
Federal Aviation Administration

Lorraine Duvall
IIT Research Institute

Sheila Frankel
National Bureau of Standards

Tony Green
Federal Trade Commission

Herbert Hecht
SoHaR Incorporated

Terry Heidelberg
Lawrence Livermore Laboratory

Larry Hoover
CRC

Raymond Houghton
National Bureau of Standards

Arnold Johnson
Federal Computer Testing Center

Linda Lawrie
US Army - CERL

Larry Lombando
Rome Air Development Center

Frank McGarry
NASA Goddard Spaceflight Center

Albert Moy
Federal Bureau of Investigations

Albrecht Neumann
National Bureau of Standards

Pat Powell
National Bureau of Standards

Carol Proctor
Illinois Institute of Technology

Wray Sexson
Defense Mapping Agency

Al Sorkowitz
Department of Housing & Urban Devlpmt.

Janet Stearns
Defense Mapping Agency

Henry G. Steubing
Naval Air Development Center

Susan Voigt
NASA Langley Research Center

REVIEWERS

Pio De Feo
NASA Ames Research Center

Patricia (Santoni) Oberndorf
Naval Ocean Systems Center

Marvin Zelkowitz
University of Maryland